SLAM BASED AIRBORNE VISUAL MAPPING SYSTEM

by

Zhaoxi Zhang

December 2022

A thesis submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York
in partial fulfilment of the requirements for the
degree of

Master of Science

Department of Electrical Engineering

I grant the State University of New York at Buffalo the non-exclusive right to use this work for the University's own purposes and to make single copies of the work available to the public on a not-for-profit basis if copies are not otherwise available.

Zhaoxi Zhang

Table of Contents

List of Figures

Abstract

The airborne visual SLAM system is a UAV carried visual SLAM system for aerial mapping. It provides a solution for large-scale blockage detection and modeling. UAV's mobility dramatically increases mapping efficiency, visual SLAM engine's support for the optical camera also simplifies the sensing procedure. We use the ORB-SLAM3 as the SLAM engine to get support for IMU data input. In addition, a heavy-duty octocopter is developed as the carrier vehicle. The sensed map then presents in pointcloud and octomap format. This sensing system is designed for providing terrain and blockage information for wireless environment simulation and UAV/robot swarm dispatching.

Chapter 1

Introduction

1.1 Motivation

With the advancement of wireless communications and UAV technologies, implementing flexible wireless networks and UAV-carried wireless stations has become possible. Autonomous UAVs can carry microwave, millimeter wave, and terahertz band base stations to form a flexible wireless network [1, 2, 3]. There are also many vehicles or robots carried deployable ground stations that can form a ground-based mobile network [4]. However, for mobile ground stations, how to dispatch the station's positions according to the blockages and the dynamically changing network load is challenging [5, 6]. To better deploy the mobile stations, many wireless electromagnetic space simulators can help us simulate and predict the dynamically changing wireless network coverage situations and estimate the effects of blockages on the network [7, 8]. However, it takes time to map the mission area and run the simulations [9]; this may slow down the deployment of mobile wireless networks. Terrain or buildings may also impede the flying safety of autonomous UAVs [10]. Therefore it is essential

to make the mapping process simple and fast for mobile wireless networks.

1.2 Challenges

Due to the limitations of the UAV's endurance and maximum take-off weight, the mapping systems' weight, size, power supply, and computing resource consumption are limited. In addition, a flying platform can provide a limited bandwidth for transmitting the map. Mapping and publishing the map in real-time is also challenging.

1. Mapping systems' running environment. UAVs are flying platforms, which means the mapping systems must be light and energy efficient. The oscillations of the flying platform may also affect the mapping systems' accuracy and precision. We use a visual SLAM system so optical cameras can be used as sensors. The SLAM engine is ORB-SLAM3 [11], the first visual SLAM engine that supports multiple cameras and acceleration data fusion, which can counter the oscillations, and the acceleration data can be acquired from either the flight controller or an independent sensor. For autonomous UAVs, its onboard computer is capable of processing the ORB-SLAM3 programs.

2. Map transmitting. After the mapping system generates a map for the ambient environment, the map must be transmitted to other autonomous vehicles. Nevertheless, since the mapping vehicle is flying in an unknown area, the wireless connection between the mapping vehicle and other vehicles may be unstable [12]. Moreover, the gathered terrain map contains lots of detailed shape information, which may take too much bandwidth

during transmission. Traditional mapping systems would use a point-cloud to represent the map. Based on the point cloud, we use the octomap format to compress the map so the map transmission bandwidth will be saved.

3. Real-Time Sensing. On traditional mapping missions, the mapping systems always need to gather all the terrain shape details, then generate and process the map offline. However, for a mobile network, its fast deployment characteristics require the mapping systems to work in real-time. To fulfill this requirement, we use the ROS environment for receiving the camera input and publishing the map in real-time.

1.3 Contributions

We aim to build a UAV that can carry a SLAM system, process all the SLAM computing onboard, and publish maps to other autonomous vehicles and wireless environment simulation systems. The map is compressed for transmission to other autonomous vehicles and sent into wireless environment simulators. Furthermore, this autonomous vehicle must reserve enough carrying capability for payloads like wireless base stations to be part of the mobile wireless network. We hope such an autonomous vehicle can enable the entire mobile wireless network to have a less deployment time and become more robust against blockage effects.

Chapter 2

Related Work

2.1 Overview

For the SLAM system, there are many sensors that can be used to map and sense the ambient environment for reconstruction. Typically, radar, ultrasound, and optical cameras can all be used as SLAM sensors. The Microsoft Kinect is a RGBD sensor based on an infrared sensor and monocular camera. For the SLAM engines, there are algorithms based on optical flow, SIFT, SURF, or FAST descriptors. The FAST descriptor compares a pixel's brightness and other pixels' brightness around it in a preset range to find out corners for tracking movements. There are also many applications of SLAM system integrated vehicles or robots. For example, commercial drones always carry stereo cameras for collision avoidance. Many vehicle manufacturers are also starting to install SLAM systems on their vehicles for lane-keeping assist, adaptive cruising control, or higher-level driving assist functions and safety features. Some vacuum robots also carry a LIDAR for home environment mapping, modeling and navigation.

2.2 ORB-SLAM

The ORB-SLAM [13] is a monocular camera-based SLAM system, and its tracking is based on extracting the ORB descriptor. This SLAM engine includes three threads: tracking, local mapping and loop closing. The tracking module extracts the ORB feature from each frame as a feature point. The feature points are compared between different frames to decode the camera's movement. Frames including feature points for tracking are marked as keyframes. In the local mapping thread, keyframes and feature points are put together to form the map, and redundant keyframes are deleted. Close loops in the map will be detected and corrected in the loop closing thread. To maintain tracking without taking too many computing resources, this SLAM engine has a "survival of the fittest" strategy: add all necessary frames as keyframes and then delete the redundant ones. This makes the engine have robust tracking performance while forming a compact map. This is the first version of the ORB-SLAM. Further developments on this SLAM engine supported stereo camera and RGBD camera input and were released as ORB-SLAM2. The latest update added IMU data fusion on this SLAM engine and has been published as ORB-SLAM3.

2.3 VINS-Mono

SLAM systems are designed to reconstruct the ambient environment in a map. Most SLAM algorithms keeps a global map. If the tracking is lost, the SLAM system will start initialization again. During this initialization procedure if the slam systems relocalizes successfully, the world map's update will be resumed. However if the relocalization did not work, the original world map will be dis-

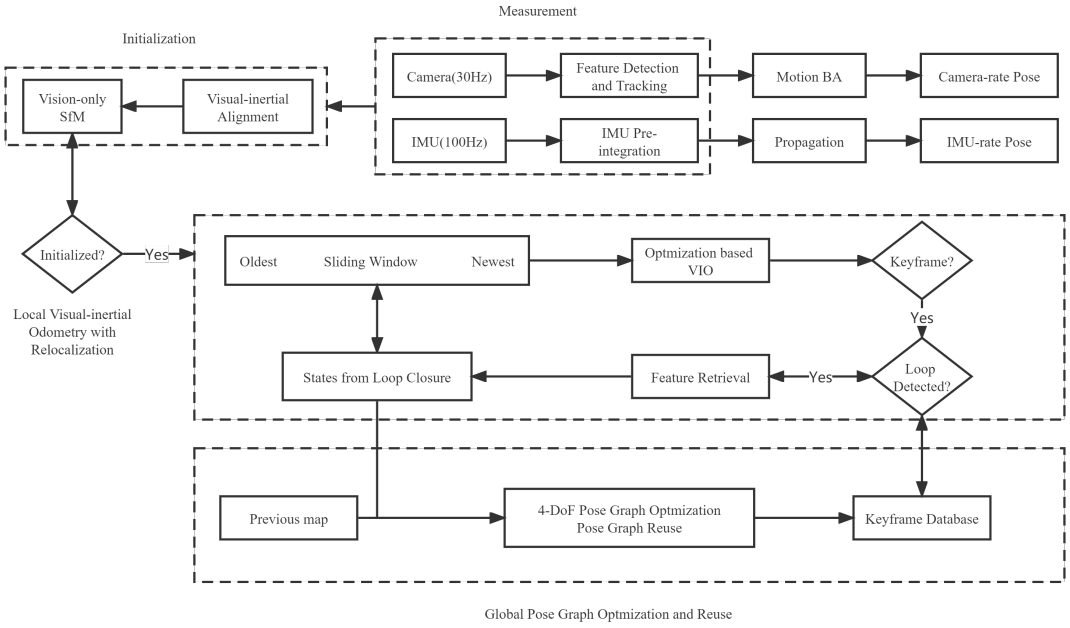

Figure 2.1: VINS-Mono processing flow graph

carded. VINS-Mono [14] is a monocular camera-IMU fusion SLAM engine with multi-map saving function. This multi-map function is based on pose graph. Multiple pose graphs can be saved. Once replicate pose graph is recognised, the pose graphs can be merged together.

Chapter 3

Mapping Techniques

A mapping system requires sensors to capture the features of the physical world to reconstruct the model of the ambient environment. A sensor like LIDAR can directly get the distance between objects and the sensor for reconstruction, while optical cameras rely on the SLAM engine to track image movements and decode distances for reconstruction. Here we introduce three popular sensing methods: ORB-feature-based tracking, infrared-based RGBD camera, and laser-based LIDAR.

3.1 Visual based mapping

Optical cameras are tiny, low cost and energy efficient sensors for acquiring images. To reconstruct a map from images, a SLAM engine is required for extracting and tracking features: extract feature points in images of the observed objects in different view angles until these feature points converge to their 3D position, then track the movements of those feature points. A state-of-the-art feature-based SLAM engine is ORB-SLAM, which uses ORB descriptors for

tracking. The latest ORB-SLAM3 [13] has supported not only the camera input but also the depth camera and inertia input. Next, we introduce the ORB-based tracking.

3.1.1 Oriented FAST Feature

The ORB (Oriented FAST and Rotated BRIEF) [15] is a scale and rotation invariant feature descriptor based on FAST (Features from Accelerated and Segments Test). The FAST locates feature points by brightness. For a pixel in an image, the FAST compares its brightness with the other 16 pixels which fall on a circle around it. If 8 pixels on this circle are darker or brighter than the center pixel, then the center pixel is marked as a feature point. The preset threshold can be changed, and the ORB uses FAST-9, which makes the threshold 9 pixels. To make FAST invariant to scale change, the ORB down-samples the original image in different scales and arranges those images as a pyramid. The FAST is run on each layer of this pyramid to detect feature points. This makes the ORB tracking continuous when the mapping camera moves closer or further to the tracked object. Around a feature point, a square of pixels is chosen as a patch. Then, ORB tracks rotation by detecting intensity changes with intensity centroid, which defines the moments of a patch as

$$m_{pq} = \sum_{x,y} x^p y^p I(x,y). \tag{3.1}$$

The centroid is then found within the moments by

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}). \tag{3.2}$$

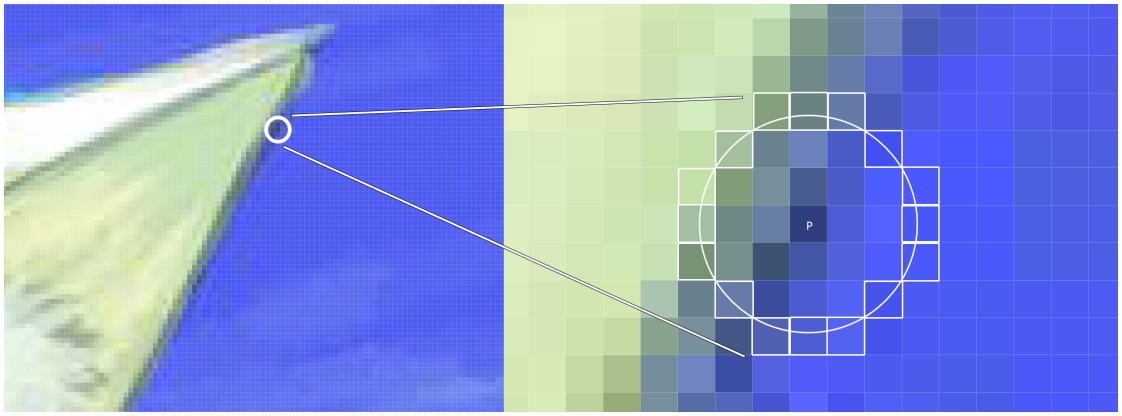

Figure 3.1: A center pixel P and the 16 pixels around it picked by FAST

Then, the direction of this patch can be described as a vector from the feature point to the centroid by

$$\theta = atan2(m_{01}, m_{10}). \tag{3.3}$$

With this direction of the patch, the ORB descriptor is partially rotation invariant.

3.1.2 Rotated BRIEF

The ORB uses BRIEF (Binary Robust Independent Elementary Feature) descriptor. The BRIEF stores feature points in a binary vector, typically 128-bit long. Firstly, the BRIEF chooses a random pixel around the feature point by Gaussian distribution, then the next pixel around the first pixel in the same way. If the first pixel is brighter than the second one, then the first bit in the binary vector is marked as 1, and 0 otherwise. This procedure repeats 128 times for a feature point to fill its 128-bit binary vector. However, BRIEF is sensitive to rotations, so ORB adds a rotation aware function for BRIEF, which makes it rBRIEF [16]. The rBRIEF defines a binary test τ with the intensity $p(x)$

$$\tau(p;x,y) = \begin{cases} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{cases} \tag{3.4}$$

The rBRIEF can then be represented by a binary vector form by n tests

$$f(n) = \sum_{1<i<n} 2^{i-1}\tau(p;x_i,y_i). \tag{3.5}$$

This makes BRIEF partially rotation invariant when the rotation is within a few degrees. When the image rotation is obvious, the BRIEF performance is poor. Here the ORB proposes steered BRIEF: in any location (x_i, y_i), the feature set of n binary tests are formed by

$$S = \begin{cases} x_1, ..., x_n \\ y_1, ..., y_n \end{cases} \tag{3.6}$$

The patch's direction θ is then used to make the matrix steered

$$S_\theta = R_\theta S \tag{3.7}$$

so the steered BRIEF can be formed by

$$g_n(p,\theta) = f_n(p)|(x_i,y_i) \in S_\theta. \tag{3.8}$$

This makes the ORB tracking continuous as long as the θ changes are consistent. Based on feature extractors like the ORB, the visual-based mapping algorithms can track movements and implement 3D reconstruction with only images input from the camera.

3.2 Depth based mapping

3D reconstruction needs to generate an accurate and precise model for the mapped environment, the distance between points in the map should be acceptable for further application. The visual-only systems can't directly get distance information from the input camera, so the visual-only reconstruction is always sparse, and unsuitable for acquiring detailed information. There are multiple solutions for getting depth information together with image input. Currently, popular depth imaging techniques include time-of-flight and structured-light sensors.

3.2.1 Time-of-Flight camera

Time-of-Flight(ToF) sensors measure distances between the camera and the objects in front of it based on counting the round trip time of light emitted from the camera system. ToF systems can get a depth map directly or by RF-modulated

Figure 3.2: Microsoft Azure Kinect, infrared ToF camera

light. A Direct ToF system includes a pair of laser emitter and receiver. Laser pulses are launched with different angles, the receiver receives the reflected

light pulses, and the distance is calculated based on the round trip time. For an RF system like in fig 3.2, it launches modulated RF light. The reflected RF light's phase shift is used to calculate the distance instead of the round trip time.

3.2.2 Structured-light camera

Structured-light systems always include a projector and multiple optical cameras. The projector projects a specific pattern. Most structured-light systems' projection patterns are straight lines. When the lines are projected on a surface, the uneven surface will make the line distorted. The cameras receive the reflected light and calculate the distance from the pattern's distortions. Multiple cameras are used as receivers to avoid the reflected light being blocked. The projected light can be emitted by a common projector or laser interference.

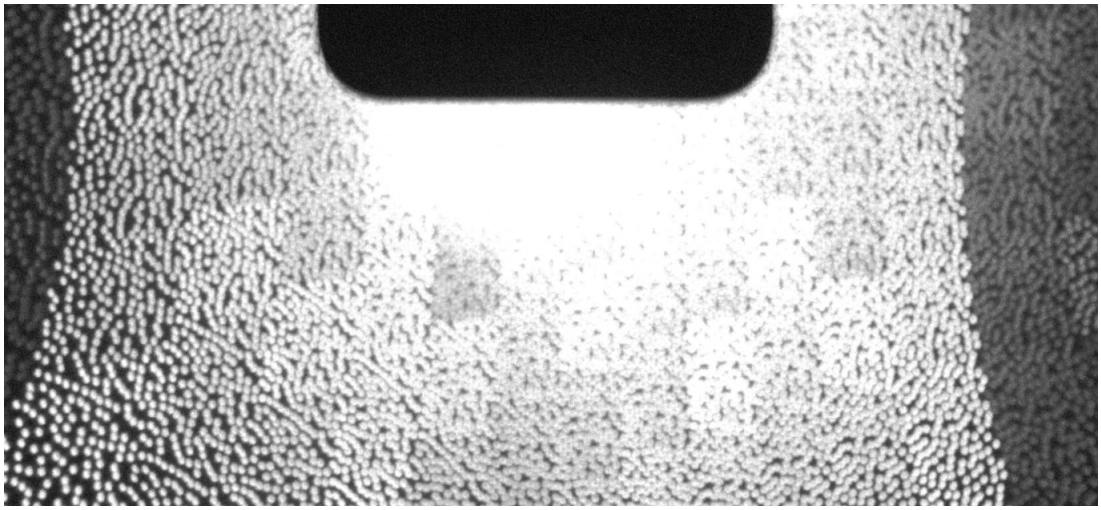

Figure 3.3: Structured-light points projected by Apple's Face ID camera

3.3 LIDAR based mapping

LIDAR is a special ToF sensor. Most LIDAR systems emit laser pulses, receive the reflected pulse and get the distance information from the round trip time. Here we mainly discuss the mechanical LIDAR. A laser pulse is launched to a high-speed spinning reflection mirror, the spinning mirror spreads laser pulses to the plane, then the 2D map of this plane can be generated [17]. Different planes map can be attached together to form a 3D map.

Figure 3.4: Velodyne Puck, a 3D LIDAR

Chapter 4

Map Process

After the mapping sensors acquire the depth feature from the environment, the sensed features and distance information will be merged together to form a map. The map generated from sensed data needs to be updated in real time and gets a correction from the SLAM engine. The map will then be optimized for further process. Next we introduce map processing techniques for handling the map.

4.1 Pointcloud

In feature-based SLAM engines, feature points will be extracted from key frames as map points. In LIDAR based scanning systems, laser impulses are launched and reflected points are received. The received raw data will be a cloud of points. Here we use the pointcloud to reconstruct the raw data from mapping system. In a pointcloud map, all the points are located by its Cartesian coordinates. To form a 3D pointcloud, we use the Point Cloud Library(PCL) [18], an open source library for handling pointcloud in different dimensions and pro-

cessing geometry data. PCL supports real-time map merging, so changes that come from sensors can be instantaneously updated. The pointcloud may include noise caused by deviations or sensor errors, so the raw pointcloud needs to minimize the noise before being used. We down-sample the pointcloud for filtering and map data compressing.

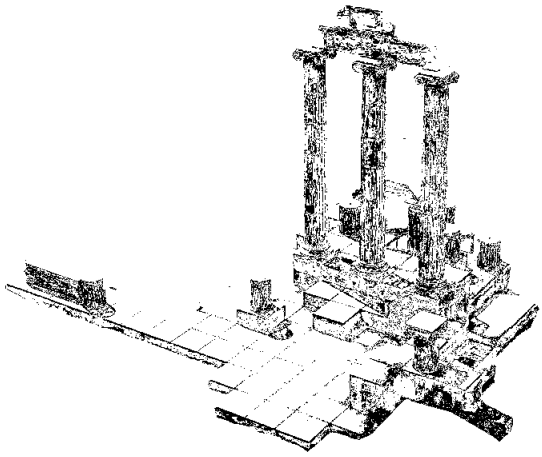

Figure 4.1: Point cloud reconstruction of UB's Baird Point

4.2 Octomap

In most 3D reconstruction applications, there are two solutions for transforming point cloud into 3D surfaces. One of them is to form triangles or Polygon meshes with vertices in the point cloud. This approach can reconstruct the details of the original surfaces more accurately. Another solution is to generate a volumetric field according to the point cloud. We will use this solution since it saves more computing resources and it is also capable of filtering out errors and deviations. The volumetric processing library we use is Octomap [19]. The

octomap is a kind of volume metric occupancy grid map. In the occupancy grid, each grid has a binary variable that represents whether it's occupied by an obstacle or is empty. The octomap generates a 3D volumetric occupancy grid according to the point cloud, occupied Greece will be represented as solid cubes, usually called a voxel. The octomap's map generation is based on octotree, a hierarchical data structure. The basic process of octotree is to fork a volumetric space into 8 sub volumetric blocks recursively until it reaches a preset resolution limitation. In a 3D space, an octotree node will only be initialized when there is an obstacle in a certain volumetric space, which saves lots of RAM when processing a map.

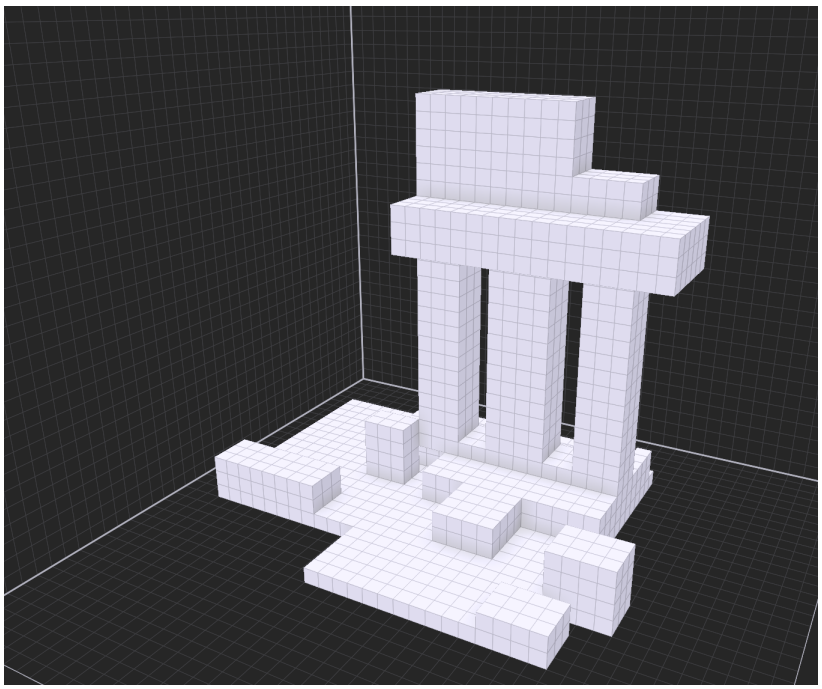

Figure 4.2: Volumetric reconstruction of UB's Baird Point based on octomap

Chapter 5

Mapping System

5.1 Overview

The mapping system consists of the input camera and lenses, the visual SLAM engine, and a map processing program. An optical camera works as an input sensor of the entire video SLAM system, and fish eye lenses give the camera a wider field of view. The SLAM engine is the core of the entire mapping system. It receives the frames input from the camera, processes the images, extracts feature points, and forms a map of the scanned environment. The map processing program extracts feature points from the SLAM engine and rebuilds the map in a point cloud format. After the point cloud is generated, the map processing program compresses it with the Octotree algorithm to save RAM and bandwidth during map transmission. The compressed Octomap represents objects in cubic blocks, making it much easier to process the map further. The mapping system will be carried on a heavy-duty multicopter for aerial mapping.

5.2 Mapping Camera

Optical cameras are convenient for large-scale, low-cost mapping. In our desired scenario, the mapping system will get an approximate contour of the terrain and buildings and provide blockage information for wireless environment simulation and UAV dispatching. For this sensing purpose, LIDAR or RGBD cameras are not the best choices since RGBD cameras' sensing range is too short, and long-range LIDARs' cost is too high. Therefore, our aerial sensing system will use monocular cameras and LIDARs. Multiple UAVs will carry a monocular mapping system and scan most of the mission area. If complicated terrain or obstacles are detected, UAVs with LIDAR can be dispatched to that area. In our monocular mapping system, cameras like Fig. 5.1 will be used as an optical sensor.

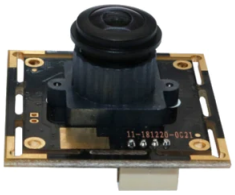

Figure 5.1: Calicam, a fisheye camera we use as monocular input

5.2.1 Camera Coordinates and Calibration

When a camera forms an image from an object, the relationship between the actual size of the object and its image is defined by four coordinate systems transformation: world, camera, image, and pixel. To reconstruct the physical world, we must set the camera intrinsic to let the SLAM engine finish the transformation between these four coordinate systems.

5.2.2 Driving camera in ROS

Our goal for this SLAM system is to acquire blockage data in real-time. Therefore, the ORB-SLAM3 engine is deployed based on the ROS environment for real-time running. In the ROS environment, we first launch the usb-cam-node with the ROS package usb-cam. Then, the node will fetch camera frames and publish them as sensor-msgs::Image in ROS topic usb-cam/image-raw. Afterward, the ORB-SLAM3 will subscribe to the published topic and run the SLAM process. However, the original ORB-SLAM engine has no pointcloud publishing function, so we have to add a pointcloud publishing node on its main thread.

5.2.3 Building camera model

As the mapping sensor, the input images must have a proper model to solve the relationship between image changes and mapping reconstruction. Optical cameras have two types: pinhole and fisheye. We will use the fisheye camera for a wider field of view. In the ORB-SLAM3 engine, the camera model is defined in the GeometricCamera class as a virtual function. The fisheye camera uses the KannalaBrandt8 model. The camera's intrinsics, distortions, and projection are defined in the model file. A SkewSymmetricMatrix is used to load the camera's intrinsic matrix. The observation uncertainty is also defined in the camera model. Once the tracking thread is running, the thread will use the ParseCamParamFile function to load the camera typesetting. First, a temporary vector vCamCalib is created to store intrinsic; then, a tracking matrix mDistCoef will keep the distortion parameter. The intrinsic will then be sent to the tracking variable mpCamera and read by the map collection function mpAtlas. Then in

the Atlas map, the camera is added and loaded. For stereo cameras, the camera modeling and loading process are similar. The model will load the second camera as the right camera and the original camera as the left camera. The tracking thread picks proper keyframes from input images to form the map.

5.3 SLAM Engine

The ORB-SLAM3 engine is the first SLAM engine for real-time mono, stereo, RGBD SLAM mapping with IMU data fusion. The ORB-SLAM3 handles input frames with three main threads as shown in Fig. 5.2. When a mapping system

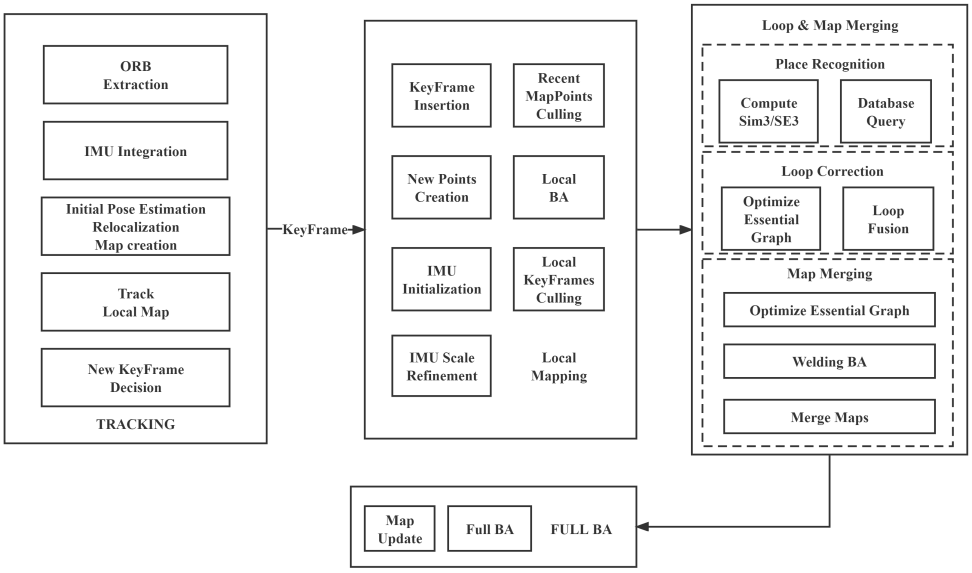

Figure 5.2: ORB-SLAM3, the SLAM engine we use to process input images

is deployed on a UAV, oscillation is inevitable and introduces deviations in the SLAM map. However, acceleration data from either the UAV flight controller or an independent IMU can erase this error. According to ORB-SLAM3's authors, this approach can get ten times more accurate than the previous version. The

visual sensor is a good solution for low cost and fast deployment for aerial mapping. In this scenario, we will use the ORB-SLAM3 engine's monocular mode for our aerial sensing.

5.4 Map Processing

The ORB-SLAM3 handles local maps with key frame culling. In feature-based mapping systems, picking how many frames as key frames is vital. Too many key frames are redundant, but too less key frames can't form a proper map. In the local mapping procedure, ORB-SLAM3 deletes redundant key frames after a local map has been formed. When processing global map, each map's Sim3/SE3 is calculated for place recognition.

5.4.1 Pointcloud publishing

Point cloud is the most common format for 3D scanning and reconstruction. Here we use a point cloud publisher to get the map from the SLAM engine and then publish it in the ROS environment. For the pointcloud publishing, we first include three ROS packages: tf, Geometry-msgs, and PCL. The tf keeps tracking all coordinates movement over frame change; the geometry-msgs transfers universal geometric data between other packages; the PCL package stores all points' x, y, and z coordinates in pointcloud. In the ORB-SLAM3 Monocular thread, we first define a position publisher and a pointcloud publisher for pointcloud publishing. Then, when the camera frames are fed into the monocular thread, we use tf to create a 3x3 matrix to store the camera frame in a right-handed coordinates system. Meanwhile, we create an empty pointcloud and

read the coordinates matrix to form the real pointcloud. Finally, the matrix and pointcloud are updated with the image grabber function in ORB-SLAM3 to ensure that the pointcloud is dynamically updated.

5.4.2 Octomap Generation

A 3D reconstruction to pointcloud can get a very detailed reconstruction of the physical world. However, our large-scale mapping does not need detailed information about obstacles shapes. Therefore, we need to process further the original point cloud acquired from the SLAM engine. Here, we use Octomap, a compact format for a 3D occupancy grid map based on octree. The Octomap represents occupied, unoccupied and unknown space. In the ROS environment, we run both the Octomap and Octovis packages, the Octomap main package handles the mapping process, and the Octovis will visualize the map.

5.5 Stereo Camera Implementation

In our initial step, we test octomap mapping with an Intel Realsense D455 camera. The D455 is a stereo camera with 6m range. We use it for indoor octomap generation and the UAV's low-height flight test. Due to the size limit of the entire sensor module, stereo camera's two sensors are close to each other, which limits its sensing range. Though stereo camera's range makes it unsuitable for UAV's onboard long range sensing, we use it for scaled mapping test to verify the octomap's process and UAV's carrying capability.

5.5.1 Indoor octomap test

To verify the performance of octomap process on UAV'S companion computer, we deploy the Octomap-server process on an Intel NUC for ground test. An Intel Realsense D455 camera is used as frame and pointcloud source. When the D455 works, it broadcasts the pointcloud yopics. We use Octomap-server to acquire it and generate the octomap.

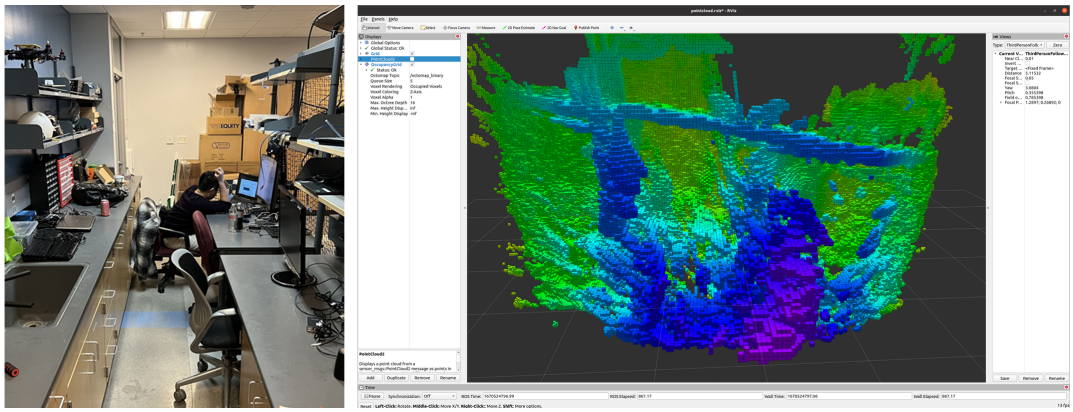

Figure 5.3: An aisle in WINGS lab and its Octomap reconstruction

We put the camera in an aisle in the WINGS lab as shown in Fig. 5.3, an occupancy map generated by octomap represents all obstacles with blocks. The color of this octomap represents depth data. The green blocks are far from the camera, and the purple ones are close. In this map, the racks in the lab are presented as blue and purple blocks, and the walls are presented as green blocks. Due to the stereo camera's range limit, the far blocks have obvious deviations.

The octomap can be updated according to the mapping area's changes. Like Fig. 5.4 shows, when a person walks into the camera's field of view, the octomap generated new blocks representing new obstacles.

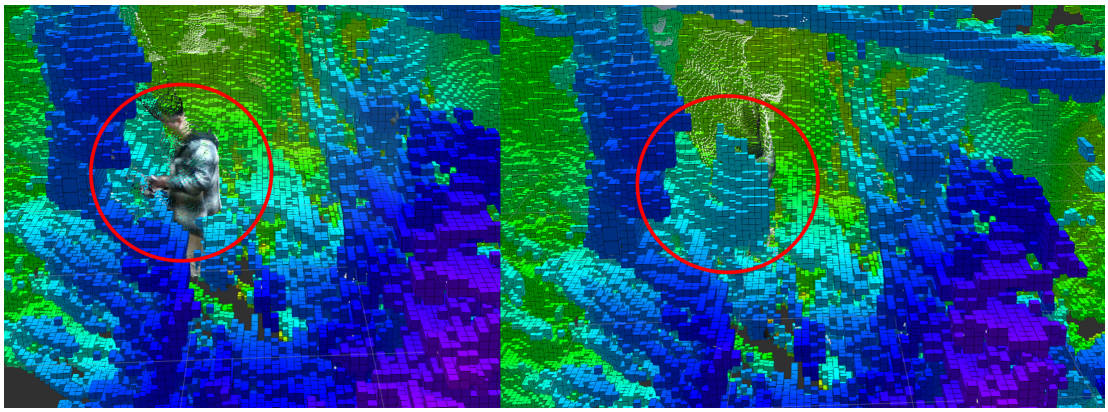

Figure 5.4: Octomap update when environment changes

Chapter 6

Carrier Vehicle

To map a mission area the mapping system must be carried by carrier vehicles. In our design, the mapping systems will be carried by flying UAVs, which are highly maneuverable for large scale mapping applications. Industrial UAS has three most popular types: fixed-wing, multicopter and VTOL. The multicopter is the easiest one to operate because it has no special requirements for taking off or landing, and therefore we will use multicopter as our carrier vehicle.

6.1 Octocopter

In our design we use an octocopter, which is a rotary-wing aircraft with eight lift-generating rotors with fixed-pitch blades. Multiple motors provide sufficient thrust for carrying our mapping payload. With the coordination of the flight controller, the octocopter is capable of maintaining flying when one of the eight motors is malfunctioning. Next we mainly introduce the power system of the Octocopter.

6.1.1 Motors

On our octocopter, we use 380 KV three phase asynchronous induction motors. The idling speed of the motor increases by 380 rpm for every volt increase in input voltage. The inertia of this motor is low and the response speed is very fast, which makes the motor itself and the copter highly maneuverable. To get the power output data, we use the Steinmetz equivalent circuit:

$$\frac{1}{s}R_r' = R_r'\frac{1-s}{s} + R_r' \tag{6.1}$$

s is slip, R_r' is rotor resistance referred to the stator.

So the output power plus motor's copper loss is the air gap power:

$$P_r = 3R_r'2I_r^{'2} \tag{6.2}$$

$$P_{gap} = \frac{3R_r'2I_r^{'2}}{s} \tag{6.3}$$

$$P_{electromechanical} = 3R_r'2I_r^{'2}\frac{1-s}{s} \tag{6.4}$$

$$P_{electromechanical} = P_{gap}(1-s) \tag{6.5}$$

P_r is rotor copper loss, I_r' is rotor current referred to the stator.

Which means the electromechanical power output with the rotation speed's relationship is

$$P_{electromechanical} = \frac{3R_r'2I_r^{'2}n_r}{sn_s}(watt) \tag{6.6}$$

n_s is synchronous speed in rpm.

According to our design, each motor consumes 237.54w and provides 1420g thrust at maximum.

6.1.2 ESC

Multicopters have fixed-pitch blades, so their maneuvering and attitude control rely on changing the motor speed rapidly, which is done by the Electronic Speed Controller (ESC). The ESC receives the signal from the flight controller and inverts the batteries' DC power into three-phase AC power to drive the motor. The ESC's MOSFET quickly switches on and off according to the flight controller's PWM signal to modulate higher or lower output power. Our ESC system is a three-phase bridge inverter with six-step control modulation. Each ESC includes three half-bridges and each half-bridge has $\frac{2\pi}{3}$ angle phase shift to generate three-phase power needed by the motor. Six-step ESC's phase voltage [20] can be expressed with Fourier series as

$$V_{ao} = \frac{2V_{DC}}{\pi}[coswt - \frac{1}{3}cos3wt + \frac{1}{5}cos5wt - ...] \tag{6.7}$$

$$V_{bo} = \frac{2V_{DC}}{\pi}[cos(wt - \frac{2\pi}{3}) - \frac{1}{3}cos(wt - \frac{2\pi}{3}) + \frac{1}{5}cos5(wt - \frac{2\pi}{3}) - ...] \tag{6.8}$$

$$V_{co} = \frac{2V_{DC}}{\pi}[cos(wt + \frac{2\pi}{3}) - \frac{1}{3}cos(wt + \frac{2\pi}{3}) + \frac{1}{5}cos5(wt + \frac{2\pi}{3}) - ...] \tag{6.9}$$

So the motor's line voltage is

$$V_{ab} = \frac{2\sqrt{3}V_{DC}}{\pi}[cos(wt + \frac{\pi}{6}) + 0 - \frac{1}{5}cos5(wt + \frac{\pi}{6}) - \frac{1}{7}cos7(wt + \frac{\pi}{6}) + ...] \tag{6.10}$$

$$V_{bc} = \frac{2\sqrt{3}V_{DC}}{\pi}[cos(wt - \frac{\pi}{2}) + 0 - \frac{1}{5}cos5(wt + \frac{\pi}{2}) - \frac{1}{7}cos7(wt - \frac{\pi}{2}) + ...] \tag{6.11}$$

$$V_{ca} = \frac{2\sqrt{3}V_{DC}}{\pi}[cos(wt + \frac{5\pi}{6}) + 0 - \frac{1}{5}cos5(wt + \frac{5\pi}{6}) - \frac{1}{7}cos7(wt + \frac{5\pi}{6}) + ...] \tag{6.12}$$

We use 22.2V LiPo battery as power system input. The ESC modulates the

input power into a proper voltage to drive the motor at a desired rpm. Among the output power, the square waves includes $6n \pm 1$ characteristic harmonics, which causes efficiency loss and motor noise. Through six-step control, we can minimize the efficiency loss caused by harmonics.

6.1.3 Flight Test

To ensure the copter is capable of mapping missions, we installed a D455 camera on it as mapping camera and set up an octomap server on the companion computer Intel NUC. As shown in Fig. 6.1, the mapping system works normally onboard. On the ground station, we can see that the ground below the UAV has been represented as blue and purple blocks.

Figure 6.1: The Octocopter with mapping system in testing facility

Chapter 7

Conclusions and Future Work

In this research, we tested ORB feature based SLAM engines and built a multi-copter. In the mapping system, we repeated monocular pinhole, Kanalabrandt8 and stereo model based camera mapping. We discussed popular SLAM and mapping techniques and camera modeling. In the carrier vehicle, we built an octocopter, tested carrying mapping payloads and discussed about the motor and its driving system.

In future works, we will install a full set of mapping system on carrier vehicles and run real mapping tests. In addition to the cameras, LIDARs will be added for long-range, high-resolution mapping. The carrier vehicles will also add different types of UAS for large-scale mapping missions. Universal octocopters will carry most mission payloads, industrial hexacopters will carry vital payloads and computers to serve as flying data centers.

Bibliography

[1] S. K. Moorthy and Z. Guan. Beam Learning in MmWave/THz-band Drone Networks Under In-Flight Mobility Uncertainties. *IEEE Transactions on Mobile Computing*, 21(6):1945–1957, June 2022.

[2] Yong Zeng, Rui Zhang, and Teng Joon Lim. Wireless communications with unmanned aerial vehicles: opportunities and challenges. *IEEE Communications Magazine*, 54(5):36–42, 2016.

[3] Sanjib Sur, Xinyu Zhang, Parameswaran Ramanathan, and Ranveer Chandra. Beamspy: Enabling robust 60 ghz links under blockage. In *USENIX NSDI*. USENIX - Advanced Computing Systems Association, April 2016.

[4] Jiangqi Hu, Sabarish Krishna Moorthy, Ankush Harindranath, Z. Guan, N. Mastronarde, E. S. Bentley, and S. Pudlewski. SwarmShare: Mobility-Resilient Spectrum Sharing for Swarm UAV Networking in the 6 GHz Band,". In *Proc. of IEEE International Conference on Sensing, Communication and Networking (SECON)*, Virtual Conference, July 2021.

[5] Lang Ruan, Jinlong Wang, Jin Chen, Yitao Xu, Yang Yang, Han Jiang, Yuli Zhang, and Yuhua Xu. Energy-efficient multi-uav coverage deployment in uav networks: A game-theoretic framework. *China Communications*, 15(10):194–209, 2018.

[6] Zhangyu Guan, Nan Cen, Tommaso Melodia, and Scott Pudlewski. Joint Power, Association and Flight Control for Massive-MIMO Self-Organizing Flying Drones. *IEEE/ACM Transactions on Networking*, 28(4):1491–1505, August 2020.

[7] J. Hu, M. McManus, S. K. Moorthy, Y. Cui, Z. Guan, N. Mastronarde, E. S. Bentley, and M. Medley. NeXT: A Software-Defined Testbed with Integrated Optimization, Simulation and Experimentation. In *Proc. of IEEE Future Networks World Forum (FNWF): WS5: Federated Testbed as a Service for*

Future Networks: Challenges the State of the Art, Montreal, Canada, October 2022.

[8] N. Mastronarde, D. Russell, Z. Guan, G. Sklivanitis, E. S. Bentley, and M. Medley. RF-SITL: A Software-in-the-loop Channel Emulator for UAV Swarm Networks. In *Proc. of IEEE WoWMoM Workshop on Wireless Networking, Planning, and Computing for UAV Swarms (SwarmNet)*, Belfast, United Kingdom, June 2022.

[9] Hailong Qin, Zehui Meng, Wei Meng, Xudong Chen, Hao Sun, Feng Lin, and Marcelo H. Ang. Autonomous exploration and mapping system using heterogeneous uavs and ugvs in gps-denied environments. *IEEE Transactions on Vehicular Technology*, 68(2):1339–1350, 2019.

[10] Ursula Challita, Walid Saad, and Christian Bettstetter. Deep reinforcement learning for interference-aware path planning of cellular-connected uavs. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, 2018.

[11] Carlos Campos, Richard Elvira, Juan J. Gomez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[12] Margarita Gapeyenko, Irem Bor-Yaliniz, Sergey Andreev, Halim Yanikomeroglu, and Yevgeni Koucheryavy. Effects of blockage in deploying mmwave drone base stations for 5g networks and beyond. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2018.

[13] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, oct 2015.

[14] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[15] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[16] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[17] Yi Lin, Juha Hyyppä, and Anttoni Jaakkola. Mini-uav-borne lidar for fine-scale mapping. *IEEE Geoscience and Remote Sensing Letters*, 8(3):426–430, 2011.

[18] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[19] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at https://octomap.github.io.

[20] J. Rodriguez, Jih-Sheng Lai, and Fang Zheng Peng. Multilevel inverters: a survey of topologies, controls, and applications. *IEEE Transactions on Industrial Electronics*, 49(4):724–738, 2002.